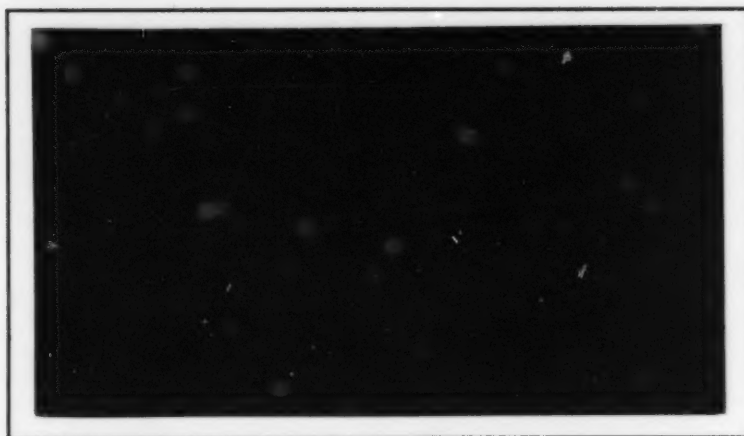




CENTRE FOR
SYSTEMS
SCIENCE

LCCR

**Automatic Generation of Compiled
Forms for Linear Recursions**

Jiawei Han and Kangsheng Zeng

CSS/LCCR TR91-06

August 1991

Automatic Generation of Compiled Forms for Linear Recursions

Jiawei Han and Kangsheng Zeng

School of Computing Science

Simon Fraser University

British Columbia, Canada V5A 1S6

{ han, zeng }@cs.sfu.ca

Abstract. *In this paper, a matrix expansion-based compilation technique is developed which compiles complex linear recursions into highly regular compiled forms. The technique uses a variable connection matrix, the V-matrix, to simulate recursion expansions and discover the expansion regularity of complex linear recursions. Our study shows that linear recursions can be compiled into highly regular compiled forms by the V-matrix expansion technique and such compiled forms can be generated automatically. The compilation of linear recursions into compiled forms not only captures the bindings which are difficult to be captured otherwise but also facilitates the development of powerful query analysis and evaluation techniques for complex linear recursions in deductive databases.*

Keywords. *Deductive databases, compilation techniques, linear recursions, and recursive query evaluation.*

1. Introduction

Compilation is a powerful preprocessing technique in the evaluation of recursions in deductive database systems [3, 4, 6-8, 10, 15, 21, 26, 28, 32]. Compilation methods can be classified into two categories, *query-dependent compilation* and *query-independent compilation*. The former rewrites a logic program into an equivalent but more efficiently evaluable one *based on specific query forms* [3, 4, 6, 31]. The Magic rule rewriting technique [3, 6, 31] is a typical such method. The latter compiles IDB predicates into a set of compiled forms *independent of queries*. When a query is submitted to the system, the compiled forms are incorporated with database statistics and query information in the query analysis and evaluation.

There have been some interesting studies on the query-independent compilation of complex linear recursions. Henschen and Naqvi [15] studied the compilation of some linear recursions in complex variable patterns. Ioannidis [16] studied the recognition of a subclass of bounded linear recursions. Sagiv [27] studied the optimization of Datalog programs. Naughton studied the compilation of one-sided linear recursions in some complex variable patterns [23]. Jagadish, Agrawal and Ness [19] studied the compilation of linear recursions based on an expansion mechanism. Agrawal and Devanbu [1] studied moving selections into complex linear recursions which involve

complex variable connections. Youn, Henschen and Han [35] classified linear recursions into several classes and discovered the expansion regularities for certain classes. Han [11] studied the compilation of complex linear recursions using a variable connection graph, the V-graph, and concluded that a linear recursion can be compiled into either a bounded recursion or an n-chain recursion. Unfortunately, none of the above studies generated an efficient algorithm to automatically generate compiled forms for linear recursions. Al-Sukairi and Henschen [2] proposed a simulation-based compilation method which applies a set of matrices to simulate the expansions of linear recursions. However, their matrix construction method is rather complex, and moreover, the method cannot discover the minimum expansion sequences for some linear recursions. In this paper, we present a variable connection matrix, the V-matrix, and develop a matrix expansion technique, which discovers the minimal necessary expansions in the compilation of complex linear recursions. Moreover, the compiled forms of linear recursions can be generated automatically based on such matrix expansions.

The motivation of our study is illustrated in the following example. As a notational convention, the syntax of PROLOG [29] is adopted for logic rules except the upper case letters are used to denote predicates and the lower case letters to denote attribute vectors, with those starting with $s, \dots, z$ indicating variables and others indicating constants.

Example 1. The recursion consisting of $\{(SG_0), (SG_1)\}$ is a typical linear recursion in chain rules [5], where SG means the same generation cousin [3, 4, 31].

$(SG_0) \quad SG(x, y) :- SIBLING(x, y).$

$(SG_1) \quad SG(x, y) :- PARENT(x, u), SG(u, v), CHILD(v, y).$

It is easy to observe by expansions that SG can be compiled into a compiled form (SG_c) [10], which consists of two *synchronous* chains, a *PARENT*-chain and a *CHILD*-chain, where *synchronous* means that the two chains grow synchronously (i.e., of the same length at each expansion).

$$(SG_c) \quad SG(x_0, y_0) = \bigcup_{i=0}^{\infty} (PARENT^i(x_{i-1}, x_i), SIBLING(x_i, y_i), CHILD^i(y_i, y_{i-1})).$$

Notice that the notation $PARENT^i(x_{i-1}, x_i)$ indicates a tautology when $i = 0$, or otherwise, a *chain* of length i which is a sequence of compositions of i *PARENT* predicates. That is,

$$PARENT^i(x_{i-1}, x_i) = \begin{cases} TRUE & \text{if } i = 0, \\ PARENT^{i-1}(x_{i-2}, x_{i-1}), PARENT(x_{i-1}, x_i) & \text{if } i > 0. \end{cases}$$

It is easy to compile the SG -recursion into a chain form because of the simplicity of the variable connections in the recursion. However, it is challenging to compile complex linear recursions, such as the recursion $\{(P_0), (P_1)\}$.

$$(P_0) \quad R(x_1, x_2, x_3, y_1, y_2, z_1, w, w) :- E(x_1, x_2, x_3, y_1, y_2, z_1, w, w).$$

$$(P_1) \quad R(x_1, x_2, x_3, y_1, y_2, z_1, w, w) :- R(x_2, x_3, x_4, y_2, y_3, z_2, u, v), \\ A(x_1, x_4), B(y_1), C(z_1, y_3, z_1), D(y_2, a, z_2), F(w, u, v, b). \quad \square$$

The goal of this paper is to develop an algorithm which automatically discovers the expansion regularity and generates highly regular compiled forms for complex linear recursions. The paper is organized as follows. In Section 2, a variable connection matrix, V-matrix, is introduced. A V-matrix may consist of one or more units depending on the variable connectivity in the matrix. The expansion regularity of the recursions with single-unit V-matrices is studied in Sections 3. The automatic generation of compiled forms for such recursions is studied in Sections 4. The expansion regularity and the generation of compiled forms for linear recursions with multiple-unit V-matrices are studied in Section 5. The efficient evaluation of the compiled forms in deductive databases is discussed in Section 6. The strength and limitations of the expansion-based compilation method are discussed in Section 7, and the study is summarized in Section 8.

2. A Variable Connection Matrix: the V-Matrix

Like many researchers, we assume that a deductive database consists of an EDB (a set of database relations) and an IDB (a set of predicates defined by function-free Horn-clause rules). A **deduction rule** is of the form " $P_0 :- P_1, P_2, \dots, P_k$," where P_i ($0 \leq i \leq k$) is of the form $P_i(x_1, \dots, x_n)$ where P_i is a predicate name, x_j ($1 \leq j \leq n$) is a variable or a constant, and j is the argument position of x_j in P_i . A rule is **recursive** if its head predicate also appears in the body (assuming there is no mutual recursion here). It is **linearly recursive** if its head predicate appears exactly once in the body. The variables appearing in the rule head are **distinguished variables**, while those appearing only in the body of the rule are **nondistinguished variables**. A **linear recursion** is a program consisting of *one linearly recursive rule* and one or more *nonrecursive (exit) rules*.

Without loss of generality, the following assumptions are made in our discussion: (1) all the nonrecursive predicates are EDB predicates; (2) there is no constant appearing in the recursive predicate (notice that a constant in a nonrecursive predicate can be eliminated in preprocessing by performing a selection using the constant and then projecting off the constant column); and (3) there is only one (*default*) nonrecursive rule (E_0) in the recursion with the head predicate $R(x_1, \dots, x_n)$.

$$(E_0) \quad R(x_1, \dots, x_n) :- E_0(x_1, \dots, x_n).$$

Definitions. For a linear recursive rule with the head predicate $R(x_1, \dots, x_n)$, the **0th expansion** of R is defined as a tautological rule,

$$R(x_1, \dots, x_n) :- R(x_1, \dots, x_n).$$

The **first expansion** of R is the recursive rule of R . The **k-th expansion** of R ($k > 1$) is the unification of the recursive rule of R with the $(k - 1)$ -st expansion of R . The **k-th expanded exit rule** of R ($k \geq 0$), denoted as $E_k(x_1, \dots, x_n)$, is the unification of the k -th

expansion with the exit rule (E_0). The **compiled form** of R is the union of all of its expanded exit rules.

The expansion behavior of a recursion is closely related to the variable connections among its predicates.

Definitions. Two predicates in the body of a rule are **connected** if they share a variable or are connected to a set of connected predicates. Two *nonrecursive* predicates in the body of a rule are **U-connected** if they share a variable or are U-connected to a set of U-connected predicates. A set of variables are **U-connected** if they are in the same nonrecursive predicate or in the same set of U-connected (nonrecursive) predicates.

The variables of a recursive rule can be partitioned based on their U-connections. In order to study the expansion behavior of a linear recursive rule, a variable connection matrix, **V-matrix**, is constructed based on the variable connections of a linear recursive rule.

Definition. The variable connection matrix, **V-matrix**, for a linear recursive rule of arity n (the arity of the head predicate) consists of a sequence of rows. Each row consists of n columns with the i -th one corresponding to the i -th argument position of the recursive predicate. Moreover, there are possibly U-connection edges between some columns in a row according to the matrix initialization and expansion rules. The **initial V-matrix**, which consists of the first two rows (row [0] and row [1]) of the matrix, is constructed according to the following V-matrix initialization rules, while the remaining rows, if any, are constructed based on the matrix expansion rules to be presented in the next section.

V-matrix initialization rules: A V-matrix is initialized in the following four steps:

- (1) Partition the variables in the rule according to the U-connections (and each partition is called a *U-connected set*);
- (2) copy the variables in the recursive predicate in the head and the body to the corresponding columns in row [0] and row [1] respectively;
- (3) replace the variable at each column of row [1], say x , by the set of distinguished variables U-connected with x , if any; and
- (4) set up a U-connection edge between each pair of columns in the corresponding row if the pair of columns are in row [0] and contain U-connected distinguished variables, or if they are in row [1] and contain U-connected nondistinguished variables.

Example 2. The initial V-matrices of the recursive rules (A_1) to (G_1) are shown in Fig. 1 (a) to (g).

- $(A_1) R(x) :- A(x, x_1), R(x_1).$
 $(B_1) R(x) :- A(x, x'), R(x_1).$
 $(C_1) R(x, y) :- A(x, y_1), R(x_1, y_1), B(x_1, y).$
 $(D_1) R(x, y, z) :- A(x, y), R(x_1, z, z_1), B(x_1, z_1).$
 $(F_1) R(x, y) :- A(x, x_1, y), B(y, y_1), R(x_1, y_1).$
 $(G_1) R(x, y, w, t, z, u, v) :-$
 $R(y, x_1, t, w_1, z_1, u_1, u_1), A(x, x_1, t), B(w, w_1), C(z), D(u, v, u_1).$

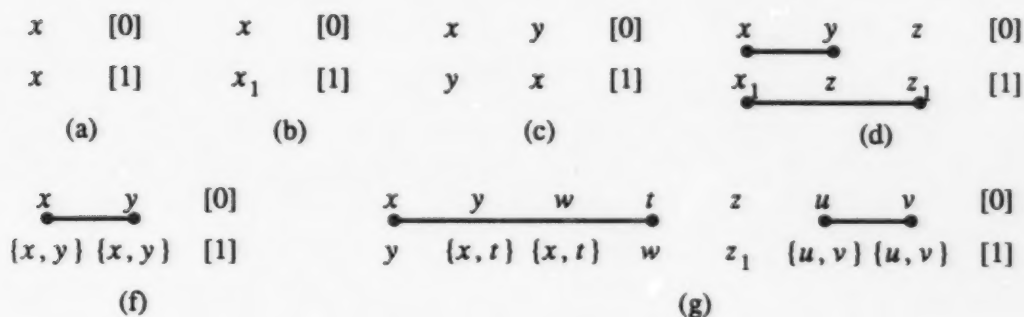


Fig. 1. The initial V-matrices of rules (A_1) to (G_1) .

For rule (A_1) , x_1 in row [1] is replaced by x as shown in Fig.1 (a) because there is a U-connected set $\{x, x_1\}$. For (B_1) , x_1 in row [1] retains as shown in Fig.1 (b) because x_1 is not in the U-connected set $\{x, x'\}$. For (C_1) , x_1 is replaced by y , and y_1 by x in row [1] as shown in Fig.1 (c) because the rule has two U-connected sets: $\{x, y_1\}$ and $\{x_1, y\}$. For (D_1) , since it has three U-connected sets $\{x, y\}$, $\{z\}$ and $\{x_1, z_1\}$, row [1] $[x_1, z, z_1]$ retains, but there are two U-connection edges, one between columns 1 and 2 in row [0] and the other between columns 1 and 3 in row [1] as shown in Fig.1 (d). For (F_1) , since there is one U-connected set $\{x, x_1, y, y_1\}$, x_1 and y_1 in row [1] should be replaced by the set of distinguished variables $\{x, y\}$, and moreover, there is one U-connection edge between columns 1 and 2 in row [0] as shown in Fig.1 (f). Finally, (G_1) contains six U-connected sets $\{x, x_1, t\}$, $\{w, w_1\}$, $\{z\}$, $\{z_1\}$, $\{y\}$ and $\{u, v, u_1\}$. Thus for the variable vector $[y, x_1, t, w_1, z_1, u_1, u_1]$ of row [1], x_1 and t are replaced by $\{x, t\}$, w_1 by w , and u_1 by $\{u, v\}$. Moreover, there are two U-connection edges in row [0], one between columns 1 and 4 and the other between columns 6 and 7 as shown in Fig.1 (g). $\square$

A V-matrix can be partitioned into one or more *unit V-matrices* based on the connections among matrix columns.

Definitions. Two columns of a V-matrix are **connected** if the two columns in the initial V-matrix share a variable, have a U-connection edge or are connected to a set of connected columns. A set of connected columns form a **unit V-matrix**. A linear recursive rule whose V-matrix consists of only one unit is a **single-unit rule**; otherwise, it is a **multiple-unit rule**.

In Ex. 2, the first five rules, (A_1) to (F_1) , are single-unit rules; while the sixth one, (G_1) , is a multiple-unit rule which consists of three units *headed by* (i.e., in row [0]) $[x, y, w, t]$, $[z]$ and $[u, v]$, respectively.

3. Expansions of Single-Unit Linear Recursive Rules

We study the expansions of corresponding V-matrices for single-unit linear recursive rules.

Example 3. The recursive rules (A_1) to (F_1) in Ex. 2 are expanded to (A_2) to (F_2) respectively in the second expansion.

$$(A_2) \quad R(x) :- A(x, x_1), A(x_1, x_2), R(x_2).$$

$$(B_2) \quad R(x) :- A(x, x'), A(x_1, x_1'), R(x_2).$$

$$(C_2) \quad R(x, y) :- A(x, y_1), B(x_2, y_1), R(x_2, y_2), A(x_1, y_2), B(x_1, y).$$

$$(D_2) \quad R(x, y, z) :- A(x, y), A(x_1, z), R(x_2, z_1, z_2), B(x_2, z_2), B(x_1, z_1).$$

$$(F_2) \quad R(x, y) :- A(x, x_1, y), B(y, y_1), A(x_1, x_2, y_1), B(y_1, y_2), R(x_2, y_2).$$

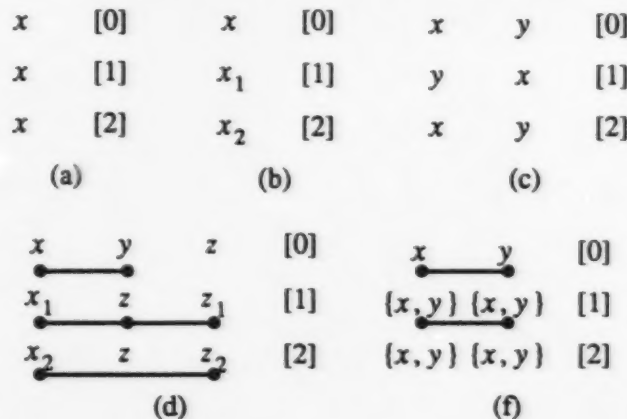


Fig. 2. V-matrices of the five rules at the second expansion.

If each recursive rule generated at the second expansion is treated as an original recursive rule, row [2] of each V-matrix can be constructed as row [1] using the same matrix initialization rules presented in Section 2. For example, row [2] in Fig. 2 (a) is $[x]$ because x_2 is in the U-connected set $\{x, x_1, x_2\}$, and x is a distinguished variable. Row [2] in Fig. 2 (b) is $[x_2]$ because x_2 is not U-connected to the distinguished variable x in (B_2) . Row [2] in Fig. 2 (c) is $[x, y]$ because there are two U-connected sets in (C_2) : $\{x, x_2, y_1\}$ and $\{x_1, y, y_2\}$, and x_2 can be replaced by x , and y_2 by y . Row [2] in Fig. 2 (d) is $[x_2, z, z_2]$ because there are three U-connected sets in (D_2) : $\{x, y\}$, $\{x_1, z, z_1\}$, and $\{x_2, z_2\}$, the second column z_1 should be replaced by the distinguished variable z , and there should be a U-connection edge between x_2 and z_2 . Finally, row [2] in Fig. 2 (e) is $[\{x, y\}, \{x, y\}]$ because there is only one U-connected set $\{x, y, x_1, y_1, x_2, y_2\}$.

Interestingly, row [2] can be derived not only from the second expansion of the recursive rule but also from the initial V-matrix, i.e., from row [0] and row [1]. For example, Fig. 2(a) indicates that if a distinguished variable x at row [0] derives the same x at row [1], it will derive the same x at row [2]. Fig. 2(b) indicates that if a distinguished variable x at row [0] derives a nondistinguished variable x_1 at row [1], it will derive a new nondistinguished variable, such as x_2 , at row [2]. Fig. 2(c) indicates that if a distinguished variable x at row [0] derives another distinguished variable y at row [1], the same x will derive the same y from row [1] to row [2]. Using the same rules observed in Fig. 2(b) and Fig. 2(c), $[x_2, z_1, z_2]$ should be obtained for row [2] in Fig. 2(d). However, since the copy of U-connection edges from row [1] to row [2] and from row [0] to row [1] make z and z_1 U-connected, z_1 in row [2] is replaced by the distinguished variable z , and row [2] becomes $[x_2, z, z_2]$. Similarly, row [2] for Fig. 2(f) is $[\{x, y\}, \{x, y\}]$ because both x and y derive the set $\{x, y\}$. $\square$

The above example shows that new rows of a V-matrix can be generated from its initial V-matrix by a set of V-matrix expansion rules, and the generated rows reflect the U-connectivities of the corresponding expanded recursive rules.

Definition. A variable y is a **derivative** of a distinguished variable x in a V-matrix if y is *derived* by x , that is, y and x is at the same column in the V-matrix but y 's row number is x 's row number + 1.

In general, the V-matrix expansion rules can be summarized as follows, where the row $\text{NewRow} (= \text{LastRow} + 1)$ is generated from the row LastRow of the V-matrix.

V-matrix expansion rules:

- (1) (*Row generation*) For each distinguished variable x in matrix $[\text{LastRow}, i]$, add x 's derivatives to matrix $[\text{NewRow}, i]$.
- (2) (*U-connection Propagation*) The U-connection edges are copied from LastRow to NewRow and then from $\text{LastRow} - 1$ to LastRow . If such copying makes a distinguished variable x U-connected to the set of variables in matrix $[\text{NewRow}, i]$, x is added to the set of variables in matrix $[\text{NewRow}, i]$.

Lemma 1. Each row of the V-matrix generated by the above two matrix expansion rules correctly registers the set of distinguished variables U-connected to each column of the recursive predicate in the body at each expansion.

Proof.

According to the matrix construction rules, the initial V-matrix registers (i) the set of distinguished variables U-connected to each argument position in the recursive predicate in the body, (ii) the U-connections among the distinguished variables in the head predicate of the recursive rule, and (iii) the U-connections among the nondistinguished variables in the recursive predicate in the body.

Suppose in the recursive rule (R_1) , x_i , the i -th variable of the head predicate, is U-connected to y_j , the j -th variables of the recursive predicate in the body. Notice that the

subscript number of a variable indicates its argument position in the recursive predicate. Suppose x_j is U-connected to y_p at the k -th expansion as shown in (R_k) . (R_k) can be rewritten to (R_k') by variable renaming (for resolution). By resolution, it is obvious that x_i is U-connected to z_p at the $(k+1)$ -st expansion, (R_{k+1}) .

$$\begin{aligned}
 (R_1) \quad & R(\dots, x_i, \dots, x_j, \dots, x_p, \dots) :- A(x_i, y_j), R(\dots, y_i, \dots, y_j, \dots, y_p, \dots), \dots \\
 (R_k) \quad & R(\dots, x_i, \dots, x_j, \dots, x_p, \dots) :- B(x_j, y_p), R(\dots, y_i, \dots, y_j, \dots, y_p, \dots), \dots \\
 (R_k') \quad & R(\dots, y_i, \dots, y_j, \dots, y_p, \dots) :- B(y_j, z_p), R(\dots, z_i, \dots, z_j, \dots, z_p, \dots), \dots \\
 (R_{k+1}) \quad & R(\dots, x_i, \dots, x_j, \dots, x_p, \dots) :- A(x_i, y_j), B(y_j, z_p), R(\dots, z_i, \dots, z_j, \dots, z_p, \dots), \dots
 \end{aligned}$$

...	x_i	...	x_j	...	x_p	...	[0]
...	y_i	...	x_i	...	y_p	...	[1]
...	...	...	...	...	...	...	...
...	...	...	...	...	x_j	...	[k]
...	...	...	...	...	x_i	...	[k+1]

Fig. 3. V-matrix expansions corresponding to rule expansions

This U-connection can be obtained by the V-matrix expansion rule (1) (Fig. 3). In the initial matrix, x_j derives x_i from row [0] to row [1]. The p -th column of row [k] contains x_j . Then the p -th column of row [k+1] should be x_i according to rule 1, which is also derivable from the expanded rule (R_{k+1}) . Similarly, if there were no expression A in (R_1) , the j -th column should be y_j in row [1]. Then the p -th column in row [k+1] should be a nondistinguished variable from the V-matrix expansion rule (1) or from the expanded rule. Similarly, other distinguished variables U-connected with x_i will play the same role. Thus the V-matrix expansion rule (1) is correct.

The correctness of the V-matrix expansion rule (2) (U-connection propagation) is proved as follows. The initial V-matrix sets up the U-connection edges for the U-connections between distinguished variables and for those between nondistinguished variables. Suppose in the initial V-matrix, the i -th and j -th (distinguished) variables in the head, x_i and x_j , are U-connected via $A(x_i, x_j)$, the nondistinguished variables in the j -th and k -th columns in the recursive predicate in the body, x_j' and x_k' , are U-connected via $B(x_j', x_k')$, and the variable at the i -th column in the recursive predicate in the body and the k -th variable in the head, x_i' and x_k , are U-connected via $C(x_k, x_i')$, as shown in (S_1) .

$$\begin{aligned}
 (S_1) \quad & R(\dots, x_i, \dots, x_j, \dots, x_k, \dots) :- \\
 & A(x_i, x_j), B(x_j', x_k'), C(x_i', x_k), R(\dots, x_i', \dots, x_j', \dots, x_k', \dots), \dots
 \end{aligned}$$

At the second expansion (S_2) , the newly set U-connections make x_k U-connected with x_i' , x_j' , x_k' and x_i'' . This verifies the correctness of the V-matrix expansion rule (2). Similarly, we can prove its correctness for other kinds of single-unit recursive rules.

$$(S_2) \quad R(\dots, x_i, \dots, x_j, \dots, x_k, \dots) :- \\ A(x_i, x_j), B(x_i', x_k'), C(x_i', x_k), A(x_i', x_j'), B(x_j'', x_k''), \\ C(x_i'', x_k'), R(\dots, x_i'', \dots, x_j'', \dots, x_k'', \dots), \dots$$

The U-connection information in the original recursive rule are fully registered in the initial V-matrix. Based on the above reasoning, we can conclude that the V-matrix expansion rules register completely the U-connection information for each column relevant to the set of distinguished variables. $\square$

In principle, a V-matrix can be expanded infinitely by following the V-matrix expansion rules. However, it is easy to observe that starting at a certain expansion, future expansions of a V-matrix will repeat the patterns of the existing rows in the V-matrix.

Definition. Two rows, row $[i]$ and row $[j]$, in a V-matrix are **identical** (denoted as $\text{row}[i] \equiv \text{row}[j]$) if each pair of their corresponding columns contains the same set (or the same empty set) of distinguished variables, where the set of distinguished variables in a column is the set of all the distinguished variables U-connected to the variable(s) in the column.

In Fig. 2, row $[1] \equiv \text{row}[0]$ in (a) and (f) (notice that two rows are identical in (f) since x and y are U-connected via a U-connection edge in row $[0]$); row $[2] \equiv \text{row}[1]$ in (b) and (d) since they have the same set or the same empty set of distinguished variables; and row $[2] \equiv \text{row}[0]$ in (c).

Clearly, once $\text{row}[j] \equiv \text{row}[i]$ ($j > i$), all the expansions starting at row $[j]$ repeats the expansions starting at row $[i]$. That is,

Lemma 2. In a single-unit V-matrix, if $\text{row}[i] \equiv \text{row}[j]$, and $i < j$, then $\text{row}[j+k] \equiv \text{row}[i+k]$ for any $k \geq 0$.

Proof.

According to the V-matrix expansion rules, all the rows in a single-unit V-matrix share the same initial V-matrix and follow the same V-matrix expansion rules in their expansions. Therefore, if two rows contain the same set of distinguished variables in their corresponding columns, their expanded rows should still contain the same set of distinguished variables in the corresponding columns. That is, if $\text{row}[i] \equiv \text{row}[j]$, then $\text{row}[i+1] \equiv \text{row}[j+1]$. By induction, $\text{row}[j+k] \equiv \text{row}[i+k]$ for any $k \geq 0$. $\square$

Based on such regularity of V-matrix expansions, the *stable level* and the *period* of a V-matrix are defined below.

Definitions. If starting at row S , there exists a T such that the row of a single-unit V-matrix repeats at every T more expansions, that is, $\text{row}[S+k \times T] \equiv \text{row}[S]$ for all $k > 0$, then S is called the **stable level** and the smallest T the **period** of the V-matrix. If row $[S]$ contains no distinguished variables, T is defined as 0.

For example, the stable levels and periods of the recursions A_1 to F_1 are as shown in Table 1.

Rule	A_1	B_1	C_1	D_1	F_1
S	0	1	0	1	0
T	1	0	2	1	1

Table 1. Stable levels and periods of the recursions A_1 to F_1 .

In general, the following algorithm is presented for the expansion of a single-unit V-matrix and the derivation of its stable level S and the period T .

Algorithm 1. The expansion of a single-unit V-matrix and the derivation of its stable level S and the period T .

Input: An initial single-unit V-matrix.

Output: An expanded V-matrix, the stable level S and the period T .

Method:

begin

 LastRow := 0; CurrentRow := 1;

while not RowRepeating (CurrentRow, ExistingRow) **do**

begin

 LastRow := CurrentRow; CurrentRow := CurrentRow + 1;

 /* Generate the contents of the CurrentRow. */

for each column i **do** /* Every column in CurrentRow is initially empty. */

for each distinguished variable x in V-matrix[LastRow, i] **do**

 Add x 's derivatives to V-matrix[CurrentRow, i];

 /* U-connection Propagation. */

 Copy the U-connections from LastRow to CurrentRow;

 Copy the U-connections from LastRow - 1 to LastRow;

for each column i **do**

for each x in V-matrix[CurrentRow, i] **do**

if x is U-connected to a distinguished variable y which is not already in V-matrix[CurrentRow, i]

then Add y to V-matrix[CurrentRow, i] and remove, if any, nondistinguished variables there

end;

S := ExistingRow;

if there is no distinguished variable in CurrentRow

then T := 0

else T := CurrentRow - ExistingRow

end.

Notice that *RowRepeating* is a Boolean function which returns true if there is an ExistingRow, where $0 \leq \text{ExistingRow} < \text{CurrentRow}$, such that $\text{row}[\text{ExistingRow}] \equiv \text{row}[\text{CurrentRow}]$. That is,

```

function RowRepeating (CurrentRow, var ExistingRow): Boolean;
begin
  ExistingRow := CurrentRow - 1;
  repeat
    If row[CurrentRow]  $\equiv$  row[ExistingRow]
    then return(true)
    else ExistingRow := ExistingRow - 1
  until ExistingRow < 0;
  return(false)
end.  $\square$ 

```

Algorithm 1 generates almost the same matrices for Ex. 3 as Fig. 2 except that it does not generate new nondistinguished variables from nondistinguished variables. That is, row [2] of Fig. 2(b) is $[\emptyset]$ instead of $[x_1]$, and row [2] of Fig. 2(d) is $[\emptyset, z, \emptyset]$ instead of $[x_2, z, z_2]$. This simplifies book-keeping effort and will not influence the derived stable levels and periods.

Theorem 1. *In a single-unit recursive rule of arity n , the expansion of its V-matrix terminates at or before the n -th iteration. That is, $S + T \leq n$.*

Proof Sketch.

The theorem can be proved by induction. The detailed proof is in the appendix with the ideas sketched here.

Since the algorithm follows the V-matrix expansion rules, it correctly derives the stable level S and the period T for both V-matrix and the recursion. We show that there exist S and T for any V-matrix, such that $\text{row}[S] \equiv \text{row}[S + T]$, and $S + T \leq n$.

Obviously, the algorithm terminates and derives $S + T = n$ if the row [0] of the V-matrix is of the form $[x_1, x_2, \dots, x_{n-1}, x_n]$, and row [1] is $[x_2, x_3, \dots, x_n, x_1]$ or $[t, x_1, x_2, \dots, x_{n-1}]$. Suppose there are additional U-connections in row [0] and/or row [1] or there are more than one distinguished variable in some column of row [1]. The role which they play is to help building additional U-connections for some columns, which creates additional chances for row repeating. Therefore, further expansions will terminate with $S + T \leq n$. $\square$

4. Automatic Generation of Compiled Forms for Single-Unit Linear Recursions

Since the variable connectivity of an expanded V-matrix corresponds to the variable connectivity of the expanded recursive rules, the regularity of the expansions of a V-matrix corresponds to the regularity of the expansions of the corresponding recursion. When a single-unit V-matrix reaches its stable level, the corresponding linear recursion reaches its *stable stage*, where a linear recursion is *stable* at the S -th expansion if (1) it

generates no new results for any EDBs by further expansions (*bounded* [24]), or (2) it adds to the body of an expanded rule the same set of EDB predicates U-connected to the same set of distinguished variables at every further T expansions (*periodicity*).

Lemma 3. *The recursion becomes stable at level S with period T , where S and T are derived from the V-matrix expansion.*

Proof.

An expanded V-matrix represents the connectivities of the distinguished variables of its corresponding expanded linear recursion. Starting at the S -th expansion, row $[S] \equiv$ row $[S + k \times T]$ ($k > 0$). This indicates that starting at the S -th expansion, the connections of the distinguished variables repeat at every further T expansions. Therefore, the recursion becomes stable at the level S with period T . $\square$

Therefore, S is also called the **stable level** of the recursion, and T the **period** of the recursion. The S -th expanded recursive rule is called the **stable rule** of the linear recursion.

Definitions. A chain of length k ($k > 1$) is a sequence of k predicates with the following properties: (1) all k predicates have the same name, say p , and the l -th p of the chain is denoted as $p_{(l)}$, (2) there is at least one shared variable in every two consecutive predicates, and if the i -th variable in the first predicate is identical with the j -th variable in the second, then (i, j) is an invariant of the chain in the sense that the i -th variable of $p_{(l)}$ is identical with the j -th variable of $p_{(l+1)}$ for every l where $1 \leq l \leq k - 1$. Each predicate of the chain is called a **chain predicate**. A chain predicate may consist of a sequence of *connected* nonrecursive predicates.

Definitions. A linear recursion is an **n -chain recursion** if for any positive integer K , there exists a k -th expansion of the recursion consisting of one chain (when $n = 1$) or n synchronous (of the same length) chains (when $n > 1$) each with the length greater than K , and possibly some other predicates which do not form a chain. It is a **single-chain recursion** when $n = 1$, or a **multi-chain recursion** when $n > 1$. A recursion is **bounded** if it is equivalent to a set of nonrecursive rules.

Lemma 4. *A single-unit linear recursion is bounded if its period $T = 0$.*

Proof.

According to the definition of the period of a V-matrix (hence a single-unit linear recursion), the period $T = 0$ indicates that at the stable level S , row $[S]$ contains no distinguished variables. Variables at the further expansion on the row are U-connected only to nondistinguished variables which have no U-connections with distinguished variables. Such U-connections cannot contribute to the new answer(s) of the distinguished variables. Thus the recursion is bounded. $\square$

For example, the recursion formed by (B_1) and the default exit rule (E_0) is bounded since its period $T = 0$.

Lemma 5. *The number of potential chains of a single-unit linear recursion is the number*

of distinct sets of connected distinguished variables in the row $[S + T]$ of its V-matrix, where S is the stable level, and T is the period of the V-matrix.

Proof.

Suppose column C of row $[S + T]$ in the V-matrix of a single-unit linear recursion contains a set of connected distinguished variables ω .

If the C -th variable v_c in the recursive predicate of the $(S + T)$ -th expansion is the same distinguished variable as the head predicate, the column generates no chains in the expansions since the distinguished variables at this column are linked to the recursive predicates directly via no nonrecursive predicates.

Otherwise, the variable v_c must be U-connected to ω via a set of nonrecursive predicates. This set of nonrecursive predicates or a portion of it will repeatedly grow at every T further expansions, which forms a chain. If column C' of row $[S + T]$ in the V-matrix contains the same set of distinguished variables ω , this column will share the same chain with the column C since the two columns are U-connected at the $(S + T)$ -th expansion. Thus, we have the Lemma. $\square$

Example 4. In Examples 2 and 3, (A_1) has one potential chain since its row [1] ($S + T = 0 + 1 = 1$) $[x]$ has one distinct set of distinguished variables; (C_1) has two potential chains since its row [2] ($S + T = 0 + 2 = 2$) $[x, y]$ has two distinct set of distinguished variables; (D_1) has one potential chain since its row [2] ($S + T = 1 + 1 = 2$) $[x_2, z, z_2]$ has one distinct set of distinguished variables; and (F_1) has one potential chain since its row [1] ($S + T = 0 + 1 = 1$) $[(x, y), \{x, y\}]$ has one distinct set of distinguished variables. $\square$

Then we analyze how to generate the chain-predicate for each potential chain. First, the set of nonrecursive predicates generated from the $(S + 1)$ -st expansion to the $(S + T)$ -th expansion forms the candidate set of the chain predicates of the compiled form. This is because the same set of predicates repeats at each T further expansions. However, there could be two situations for each column in the recursive predicate in the body of the $(S + T)$ -th expansion: (i) U-connected to a set of distinguished variables, and (ii) not U-connected to a set of distinguished variables. The predicates in the candidate set which are linked to those variables *not* U-connected to a set of distinguished variables (i.e., in situation (ii)) cannot be a component of the chain, and therefore should be replaced by their corresponding nonrecursive predicates in the previous round of expansions (i.e., from the $(S - T + 1)$ -st expansion to S -th expansion). The replaced predicates together with the remaining candidate set forms the primitive set which generates all the chain predicates. Each chain predicate represents a set of predicates in the primitive set connected to a set of columns in the recursive predicate at the $(S + T)$ -th expansion. The variables connected to the recursive predicate are indexed by i ; the distinguished variables or those which are connected to the nonrecursive predicates not in the primitive set are indexed by $i - 1$. The remaining variables are not connected to any predicate outside of the primitive set and can be ignored if the set of predicates is to be represented as one chain predicate.

Therefore, we have the following rule for generating chain predicates.

Chain generation rules.

The generation of chain-predicates for nonnull chains from the recursive rules in the $(S + T)$ -th expansion consists of the following three steps:

- (1) Find the primitive set of nonrecursive predicates, which is a set of nonrecursive predicates generated between $(S + 1)$ -st to $(S + T)$ -th expansions with the predicates not U-connected to distinguished variables replaced by its corresponding predicates in previous round of expansions.
- (2) Find the connection variable vectors for each chain predicate. Each variable shared with the recursive predicate in the body is indexed with i , and each variable which is a distinguished variable or shared with the nonrecursive predicate outside of the primitive set is indexed with $i - 1$.
- (3) Update the variables to the same updated ones for those which were the same variable but are outside of the primitive set.

Example 5. In Examples 2 and 3, the chain predicate for (A_1) is $A(x_{i-1}, x_i)$ since $A(x, x_1)$ is the only nonrecursive predicate at the first expansion ($S + T = 0 + 1 = 1$), x_1 is connected to the recursive predicate and x is a distinguished variable.

The two chains for (C_1) are " $AB(x_{i-1}, x_i)$ " and " $BA(y_{i-1}, y_i)$ " where " $AB(x, x_1) :- A(x, t), B(x_1, t).$ " and " $BA(y, y_1) :- A(t, y_1), B(t, y).$ ". This is because at the second expansion ($S + T = 0 + 2 = 2$), " $A(x, y_1), B(x_2, y_1)$ " is connected to the recursive predicate by x_2 (which is then indexed as x_i), x is a distinguished variable (which is then indexed as x_{i-1}), and y_1 can be ignored in the chain predicate AB . Similarly for the BA -chain.

The chain for (D_1) is $AB(z_{i-1}, z_i)$ where " $AB(z, z_1) :- A(t, z), B(t, z_1).$ ". This is because at the second expansion ($S + T = 1 + 1 = 2$), the candidate set for the chain predicates is " $A(x_1, z), B(x_2, z_2)$ ". However, $B(x_2, z_2)$ is not U-connected to distinguished variables in (D_2) and should be replaced by the corresponding B -predicate at the first expansion, which is $B(x_1, z_1)$. Thus the primitive set of chain predicate is " $A(x_1, z), B(x_1, z_1)$ ". Since z_1 is at the recursive predicate and thus indexed as z_i , z is a distinguished variable and thus indexed as z_{i-1} , and the other variable x_1 is ignored in the chain predicate AB .

The chain for (F_1) is $AB(x_{i-1}, y_{i-1}, x_i, y_i)$ where " $AB(x, y, x_1, y_1) :- A(x, x_1, y), B(y, y_1).$ ". This is because at the first expansion ($S + T = 0 + 1 = 1$), the candidate (and also primitive) set for the chain predicate is " $A(x, x_1, y), B(y, y_1)$ ". Since x_1 and y_1 are connected to the recursive predicate and thus indexed as x_i and y_i , and x and y are distinguished variables and thus indexed as x_{i-1} and y_{i-1} , we have the above chain predicate. $\square$

According to the above discussion, the compiled form of a single-unit linear recursion can be generated automatically by extracting chain predicate(s) from the $(S + T)$ -th expansion. The algorithm is as follows.

Algorithm 2. Generation of the compiled form for a single-unit linear recursion.

Input. A linear recursion R , its stable level S and period T .

Output. The compiled form of the recursion.

Method.

Case 1: $T = 0$. The recursion is bounded and the compiled form is the union of the expanded exit rules from 0-th to S -th expansions. That is,

$$R(x_1, x_2, \dots, x_n) = E_0(x_1, x_2, \dots, x_n) \cup E_1(x_1, x_2, \dots, x_n) \cup \dots \cup E_S(x_1, x_2, \dots, x_n).$$

Case 2: $T \neq 0$. The compiled form for the recursion can be generated as follows.

If a recursion contains only null chain predicates, it is bounded and its compiled form is the union of the k -th expanded exit rules for $0 \leq k \leq S + T - 1$. That is,

$$R(x_1, x_2, \dots, x_n) = E_0(x_1, x_2, \dots, x_n) \cup E_1(x_1, x_2, \dots, x_n) \cup \dots \cup E_{S+T-1}(x_1, x_2, \dots, x_n).$$

Otherwise, the recursion is a single- or multiple- chain recursion with the following compiled form:

$$R = SS \cup \left(\bigcup_{i=0}^{\infty} (MM, CC^i, TT) \right),$$

which consists of four portions: S -portion SS , T -portion TT , chain-portion CC^i , and misc(miscellaneous)-portion MM . The S -portion represents the rule expansion before reaching its stable stage, which is the union of E_k for k from 0 to $S-1$ if $S > 0$ or empty otherwise. That is,

$$SS = \bigcup_{i=0}^{S-1} E_i(x_1, x_2, \dots, x_n).$$

The T -portion consists of the bodies of the exit rules contributing to the period of the recursion, which is formed by the union from E_0 to E_{T-1} . That is,

$$TT = \bigcup_{j=0}^{T-1} E_j(\dots, u_i, \dots, x_i, \dots).$$

The chain-portion CC^i consists of a set of nonnull chains in the exponential form with the same exponent i . Each nonnull chain predicate is generated following the chain-generating rules and is in the form of $A(x_{i-1}, x_i)$, where A is the chain predicate, and x_{i-1} and x_i are connection variable vectors.

Finally, the miscellaneous portion, if any, is composed of the predicate(s) left in the $(S + T)$ -th expansion, i.e., those not used in the formation of the chain predicate(s).

□

Theorem 2. Algorithm 2 generates correct compiled forms for single-unit linear recursions.

Proof.

From Lemma 4, if $T = 0$, the recursion is bounded. Its $(S + 1)$ -st expanded rule is absorbed by the S -th expansion. Obviously, the union of the expanded exit rule from 0th to S -th expansion generated by Algorithm 2 is the correct compiled form.

In the case of $T \neq 0$, if the set of the variables at the $(S + T)$ -th expansion is the same as the set of distinguished variables of the head predicate, the corresponding potential chain is trivially *true*, which is the *null chain-predicate*. If a recursion contains only null chain predicates, the recursion must be bounded. Thus the formula generated by Algorithm 2 is correct.

Otherwise, it is easy to show for any integer $K \geq 0$, the K -th expanded exit rule is included in the compiled form generated by Algorithm 2. When $0 \leq K < S$, the K -th expanded exit rule is in the S -portion, SS . When $K > S$, let $j = (K - S) \text{ div } T$, and $h = (K - S) \text{ mod } T$ ($0 \leq h \leq T - 1$). The term (CC^j, E_h, MM) in the compiled form is the K -th expanded exit rule where E_h is in the T -portion TT , and the chain portion is CC^j with the period T (i.e., every $(j \times T)$ expansions form a new chain). The predicates which do not form part of the chain, if any, form the miscellaneous portion, MM . The misc-portion will not grow along the expansions and thus should not be in the exponential form. Since all the generated predicates are included in the compiled form, and the formula does not contain those not generated by the expansions, the theorem is proved. $\square$

Example 6. According to Algorithm 2, the recursions (A_1) through (F_1) in Ex. 2 generate the following compiled forms.

For (A_1) , $S = 0$ and $T = 1$. Its S -portion is ϕ , its T -portion is " $E_0(x_i)$ ", and its misc-portion is trivially true. It has one potential chain with the chain-predicate " $A(x_{i-1}, x_i)$ ". Therefore, its compiled form is (A_c) .

$$(A_c) \quad R(x_0) = \bigcup_{i=0}^{\infty} (A^i(x_{i-1}, x_i), E_0(x_i)).$$

For (B_1) , $S = 1$ and $T = 0$. The recursion is bounded, and the compiled form is (B_c) , the union of the exit rule and the first expanded exit rule.

$$(B_c) \quad R(x) = E_0(x) \cup E_1(x) = E_0(x) \cup (A(x, x'), E_0(x_1)).$$

For (C_1) , $S = 0$ and $T = 2$. Its S -portion is ϕ , its T -portion is " $E_0(x_i, y_i) \cup E_1(x_i, y_i)$ ", and its misc-portion is trivially true. It has two chains with the chain-predicates extracted from (C_2) , i.e., " $AB(x, x_1) :- A(x, t), B(x_1, t)$," and " $BA(y, y_1) :- A(t, y_1), B(t, y)$ ". The compiled form is (C_c) .

$$(C_c) \quad R(x_0, y_0) = \bigcup_{i=0}^{\infty} (AB^i(x_{i-1}, x_i), BA^i(y_{i-1}, y_i), (E_0(x_i, y_i) \cup E_1(x_i, y_i))).$$

For (D_1) , $S = 1$ and $T = 1$. Its S -portion is " $E_0(x, y, z)$ ", its T -portion is " $E_0(u, z_i, v)$ ". It has one chain with the chain predicate,

" $AB(z_0, z_1) :- A(t, z_0), B(t, z_1)$ ". Its misc-portion is " $A(x, y), B(u, v)$ ". Thus the compiled form is (D_c) .

$$(D_c) \quad R(x, y, z) = E_0(x, y, z) \cup \bigcup_{i=0}^{\infty} (A(x, y), B(u, v), AB^i(z_{i-1}, z_i), E_0(u, z_i, v)).$$

For (F_1) , $S = 0$ and $T = 1$. Its S -portion is ϕ , its T -portion is " $E_0(x_i, y_i)$ ", and its misc-portion is trivially true. It has one chain with the chain predicate " $AB(x, y, x_1, y_1) :- A(x, x_1, y), B(y, y_1)$ ". Thus the compiled form is (F_c) .

$$(F_c) \quad R(x_0, y_0) = \bigcup_{i=0}^{\infty} (AB^i(x_{i-1}, y_{i-1}, x_i, y_i), E_0(x_i, y_i)). \quad \square$$

5. Compilation of Multiple-Unit Linear Recursions

The V-matrix expansion technique for compiling single-unit linear recursions can be generalized to multiple-unit linear recursions.

Although the matrix expansion rules and the matrix expansion algorithm for single-unit linear recursions can be applied directly to multiple-unit linear recursions, it may lead to a relatively large number of expansions. Suppose there are k units in the recursion, the minimum number of expansions to reach row repeating stage in the V-matrix should be the least common multiplier of the arity of each unit, that is, $lcm(n_1, n_2, \dots, n_k)$, where n_j (for j from 1 to k) is the arity of the j -th units. For example, if a V-matrix consists of three unit matrices V_1, V_2 and V_3 , with $S_1 = S_2 = S_3 = 0, T_1 = 5, T_2 = 6$, and $T_3 = 7$, it will take $T = lcm(5, 6, 7) = 210$ expansions to find a repeating row in the combined V-matrix. Obviously, this is undesirable. However, since each V-matrix unit reaches its own stable state independent of other units, the stability of a recursion can be determined by the examination of each unit independent of others. For this example, if the three unit V-matrices are expanded separately, it will take only 7 expansions to detect the regularity of the expansion patterns and compile the multiple-unit rule.

Definition. A linear recursion whose recursive rule consists of multiple unit V-matrices reaches a **stable stage** at the S -th expansion if for every unit V-matrix V_i , either it cannot generate any new results for this unit on any EDBs (i.e., *bounded* for this unit), or every further T_i expansions add to the body of the rule the same set of EDB predicates connected to the same set of variables in the unit.

Example 7. The expansion regularity of the recursion with the recursive rule (G_1) of Ex. 2 can be obtained by the examination of each V-matrix unit. The V-matrix consists of three units, V_1 headed by (i.e., in row [0]) $[x, y, w, t]$, V_2 by $[z]$ and V_3 by $[u, v]$. According to Algorithm 1, V_1 has $S_1 = 1$ and $T_1 = 2$; V_2 has $S_2 = 1$ and $T_2 = 0$; and V_3 has $S_3 = 0$ and $T_3 = 1$ as shown in Fig. 4. Therefore, their combined stable level should be $S = \text{maximum}(S_1, S_2, S_3) = 1$, with $T_1 = 2$; $T_2 = 0$; and $T_3 = 1$. $\square$

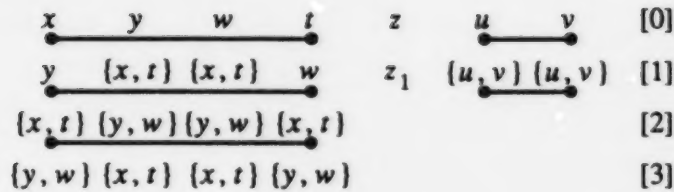


Fig. 4. The expansions of the V-matrix of rule (G_1) .

In general, the algorithm for the expansion of a multiple-unit V-matrix is summarized below.

Algorithm 3. The expansions of a multiple-unit V-matrix.

Input. An initial V-matrix V which is partitioned into k unit V-matrices, $V_1, \dots, V_k$.

Output. A stable level S of the V-matrix and the period T_i ($1 \leq i \leq k$) for each unit V-matrix V_i .

Method.

For each unit matrix V_i , derive its S_i and T_i based on Algorithm 1. Then $S = \text{maximum}(S_1, \dots, S_k)$, and each unit V_i maintains its own period T_i . $\square$

Theorem 3. If the V-matrix of a multiple-unit recursive rule consists of k unit matrices, with the i -th matrix having the stable level S_i and the period T_i , then the expansion becomes stable at the S -th expansion where $S = \text{maximum}(S_1, \dots, S_k)$, and each unit V_i maintains its own period T_i .

Proof.

Suppose $S_i < S_j$, where S_i is the stable level of the i -th unit-matrix V_i , and S_j is that of the j -th unit-matrix, V_j . At the i -th expansion, V_i stabilizes. For V_i , we have $\text{row}[S_i] = \text{row}[S_i + T_i]$. According to Lemma 2, $\text{row}[S_i + (S_j - S_i)] = \text{row}[S_i + T_i + (S_j - S_i)]$. That is, for V_i , $\text{row}[S_j] = \text{row}[S_j + T_i]$. Therefore, at the j -th expansion, V_i still repeats at every T_i further expansions. At the $S = \text{maximum}(S_1, \dots, S_k)$, every unit V-matrix repeats with its corresponding period. Thus we have the theorem. $\square$

From theorems 1 and 3, we have

Corollary. In a multiple-unit linear recursion, the number of expansions to reach the stable stage is less than the arity of its recursive predicate.

Similarly, compiled forms can be generated for multiple-unit linear recursions. We examine an example.

Example 8. In recursion G_1 , $S = 1$, $T_1 = 2$, $T_2 = 0$, and $T_3 = 1$. G_1 can be viewed as if there were three independent single-unit linear recursions: G_{11} , G_{12} and G_{13} .

$$(G_{11}) \quad R_1(x, y, w, t) :- R_1(y, x_1, t, w_1), A(x, x_1, t), B(w, w_1).$$

$$(G_{12}) \quad R_2(z) :- R_2(z_1), C(z).$$

$$(G_{13}) \quad R_3(u, v) :- R_3(u_1, u_1), D(u, v, u_1).$$

Each unit is compiled independently. For G_{11} with the header " $R_1(x, y_0, w, t_0)$ ", it has two chains with chain predicates, " $AB(y_{i-1}, w_{i-1}, y_i, w_i)$ " and " $BA(s_{i-1}, t_{i-1}, s_i, t_i)$ " respectively, where

$$AB(y_1, w_1, y_2, w_2) :- A(y_1, y_2, w_1), B(w_1, w_2).$$

$$BA(s_1, t_1, s_2, t_2) :- A(s_1, s_2, t_2), B(t_1, t_2).$$

Its T -portion is " $E_0(y_i, s_i, t_i, w_i) \cup E_1(y_i, s_i, t_i, w_i)$ ", its S -portion is " $E_0(x, y_0, w, t_0)$ ", and its misc-portion is " $A(x, s_0, t_0), B(w, w_0)$ ". Therefore, its compiled form is G_{1c} .

$$(G_{1c}) \quad R_1(x, y_0, w, t_0) :- E_0(x, y_0, w, t_0) \cup \bigcup_{i=0}^{\infty} (A(x, s_0, t_0), B(w, w_0),$$

$$AB^i(y_{i-1}, w_{i-1}, y_i, w_i), BA^i(s_{i-1}, t_{i-1}, s_i, t_i), (E_0(y_i, s_i, t_i, w_i) \cup E_1(y_i, s_i, t_i, w_i))).$$

For G_{12} , it is a bounded recursion with $R_2 = E_0 \cup E_1$ as shown in (G_{2c}) .

$$(G_{2c}) \quad R_2(z) = E_0(z) \cup E_0(z_1), C(z).$$

For G_{13} , it has one chain with the chain predicate " $D'(u_{i-1}, v_{i-1}, u_i, v_i)$ " where

$$D'(u_{i-1}, v_{i-1}, u_i, v_i) :- D(u_{i-1}, v_{i-1}, u_i), (v_i = u_i).$$

Notice that the chain predicate contains two distinguished variables. However, the predicate D consists of only three arguments. Thus an auxiliary variable v_i is introduced to the predicate with the constraint " $v_i = u_i$ ". Moreover, its T -portion is " $E_0(u_i, v_i)$ ", S -portion is ϕ , and its misc-portion is trivially true. Therefore, its compiled form is (G_{3c}) .

$$(G_{3c}) \quad R_3(u_0, v_0) = \bigcup_{i=0}^{\infty} (D'^i(u_{i-1}, v_{i-1}, u_i, v_i), E_0(u_i, v_i)).$$

The general compiled form for recursion G_1 can be derived by merging the three independent compiled forms. Since $S = 1$, $T_1 = 2$, $T_2 = 0$, and $T_3 = 1$, all the independent compiled forms should be aligned to a common S and T in order to generate a combined compiled form. Clearly, S should be the common S , and T should be the least common multiplier of the nonzero T_i 's (for i from 1 to the number of units). For this example, $T = \text{lcm}(2, 1) = 2$. (G_{1c}) is in the aligned form already. (G_{2c}) is a bounded recursion, which is the same form for any expansion greater than S . (G_{3c}) is aligned to (G'_{3c}) .

$$(G'_{3c}) \quad R_3(u, v) = E_0(u, v) \cup$$

$$\bigcup_{i=0}^{\infty} (D'(u, v, u_0, v_0), DD^i(u_{i-1}, v_{i-1}, u_i, v_i), (E_0(u_i, v_i) \cup E_1(u_i, v_i))),$$

$$\text{where } DD(u_{i-1}, v_{i-1}, u_i, v_i) :- D'(u_{i-1}, v_{i-1}, s, t), D'(s, t, u_i, v_i).$$

Therefore, the combined compiled form for recursion G_1 should be (G_c) , where the S -portion is E_0 , the T -portion is $E_0 \cup E_1$, the chain portion consists of AB -chain, BA -chain and DD -chain, and the misc-portion consists of " $A(x, s_0, t_0), B(y, y_0), C(z), D'(u, v, u_0, v_0)$ ". $\square$

$$(G_c) \quad R(x, y_0, w, t_0, z, u, v) = E_0(x, y_0, w, t_0, z, u, v) \cup$$

$$\bigcup_{i=0}^{\infty} (A(x, s_0, t_0), B(w, w_0), C(z), AB^i(y_{i-1}, w_{i-1}, y_i, w_i), BA^i(s_{i-1}, t_{i-1}, s_i, t_i),$$

$$D'(u, v, u_0, v_0), DD^i(u_{i-1}, v_{i-1}, u_i, v_i),$$

$$(E_0(y_i, s_i, t_i, w_i, z_1, u_i, v_i) \cup E_1(y_i, s_i, t_i, w_i, z_2, u_i, v_i))).$$

In general, we have

Algorithm 4. Generation of the compiled form for a multi-unit linear recursion.

Input. A multiple-unit linear recursion R , its stable level S and the period T_i ($1 \leq i \leq k$) for each unit V_i .

Output. The compiled form of the recursion.

Method.

Step 1. For each unit V_i , generate its compiled form R_i according to Algorithm 1;

Step 2. Generate the aligned compiled form for each unit V_i based on the common stable level S and the common period T , where $S = \text{maximum}(S_1, \dots, S_k)$ and $T = \text{lcm}(T_1, \dots, T_k)$;

Step 3. Merge the multiple aligned compiled forms into one combined compiled form in which

(i) the S -portion consists of the union of E_0 to E_{S-1} if $S > 0$ or empty otherwise;

(ii) the T -portion consists of the union of E_i 's for i from 0 to $T - 1$;

(iii) the chain-portion consists of all the nonnull chains, with each chain predicate determined within its unit and then aligned up for merging. All the chain predicates are in the exponential form with the same exponent i , and each variable connected to the set of distinguished variables is in the form of x_{i-1} for a distinct x , and that connected to the set of variables in the recursive predicate is in the form of x_i ; and

(iv) the misc-portion consists of the predicates at the $(S + T)$ -th expansion which does not participate the chain predicates. $\square$

Theorem 4. Algorithm 4 generates correct compiled forms for multiple-unit linear

recursions.

Proof.

Step 1, the generation of compiled form for each component unit, follows Algorithm 2. Its correctness has been proved by Theorem 2. The correctness of Step 2, the generation of the aligned compiled form for each unit based on the common stable level S and the common period T is correct since $S = \text{maximum}(S_1, \dots, S_k)$, $T = \text{lcm}(T_1, \dots, T_k)$, and the compiled form simply rewrites its original form by different grouping. Step 3 merges the multiple aligned compiled forms into one combined compiled form. Its S -portion consists of the union of all the S portions up to the common stable level S . Its T -portion consists of the union of E_i 's for i from 0 to $T - 1$. Its chain-portion consists of all the nonnull chains, and its misc-portion consists of the predicates at the $(S + T)$ -th expansion. Therefore, the algorithm generates correct compiled forms for multiple-unit linear recursions. $\square$

6. Evaluation of compiled recursions

The compilation of complex linear recursions into highly regular compiled forms facilitates query analysis and evaluation. The compiled forms can be categorized into *bounded* and *n-chain* forms. A *bounded form* consists of a finite set of relational algebra expressions; and an *n-chain* form consists of one or more highly regular compiled chains with infinite length.

A bounded recursion can be evaluated by nonrecursive (relational) query evaluation methods [22]. A single-chain recursion can be evaluated by a typical partial (query-relevant) transitive closure algorithm, such as [17, 20]. A multi-chain recursion can be evaluated by Counting, Magic Sets [3], and other proposed methods, such as [7, 15, 21, 34]. Moreover, refinement techniques such as existence checking [10], chain-split evaluation [13], constraint-based reasoning [14], etc. can also be applied based on the compiled form, the query and the database statistics.

The major benefit of the expansion-based compilation is that it generates highly regular compiled forms which provide precise chain connection information in relational expressions and facilitate the derivation of flexible and efficient processing plans for complex queries. By quantitative analyses of compiled chain forms, it is straightforward to determine the start end of the chain, the evaluation directions, the possible existence checking evaluation [10], the pushing of query constraints at the start end of a chain, and the pushing of other constraints into the chain for efficient evaluation [14]. Therefore, compilation is a powerful preprocessing technique which leads to efficient evaluation of complex queries in deductive databases.

Alternatively, the matrix-based compilation can also be viewed as a *query-independent rule rewriting* process, which transforms a complex linear recursion into a bounded recursion or a normalized *n-chain* recursion, where a normalized *n-chain* recursion is defined as follows.

Definition. A normalized *n-chain* recursion is a recursion consists of an exit rule and

a normalized n -chain recursive rule in the form of (N_1) .

(N_1) $R(x_1, x_2, \dots, x_n, w) :- C_1(x_1, y_1), C_2(x_2, y_2), \dots, C_n(x_n, y_n), E(y_1, y_2, \dots, y_n, w)$.
where x_i, y_i (for $1 \leq i \leq n$) and w are variable vectors, and C_i 's are chain predicates.

A normalized n -chain recursion provides a highly regular recursive rule for the exploration of various kinds of efficient evaluation techniques. Two examples are presented on the normalization of complex linear recursions by query-independent rule rewriting and the efficient evaluation of normalized recursions.

Example 9. Suppose a query " $?-R(a, y, c)$." is posed to the recursion $\{(D_0), (D_1)\}$ of Ex. 2 whose compiled form is (D_c) (Ex. 6).

$$(D_c) \quad R(x, y, z_0) = E_0(x, y, z_0) \cup \bigcup_{i=0}^{\infty} (A(x, y), B(u, v), AB^i(z_{i-1}, z_i), E_0(u, z_i, v)),$$

The recursion is equivalent to the following set of rewritten rules, where the only recursive rule (T_1) is a normalized 1-chain rule.

$$(D_0) \quad R(x, y, z) :- E_0(x, y, z).$$

$$(D_1) \quad R(x, y, z) :- A(x, y), T(z).$$

$$(T_0) \quad T(z) :- E_0(u, z, v), B(u, v).$$

$$(T_1) \quad T(z) :- AB(z, t), T(t).$$

The constants a and c in the query " $?-R(a, y, c)$." can be easily pushed into the rule (D_1) as " $A(a, y), T(c)$." Since $T(c)$ is a *closed* query (with the answer *yes* or *no*), it can be evaluated by an existence-checking parital transitive closure algorithm [10], that is, the evaluation terminates the first time when $T(c)$ is evaluated to *true*. $\square$

Example 10. Suppose a query " $?-R(a, b, y, d, e, u, v)$." is posed to the recursion $\{(G_0), (G_1)\}$ of Ex. 2 whose compiled form is (G_c) .

$$(G_c) \quad R(x, y_0, w, t_0, z, u, v) = E_0(x, y_0, w, t_0, z, u, v) \cup \bigcup_{i=0}^{\infty} (A(x, s_0, t_0), B(w, w_0), \\ C(z), AB^i(y_{i-1}, w_{i-1}, y_i, w_i), BA^i(s_{i-1}, t_{i-1}, s_i, t_i), D'(u, v, u_0, v_0), \\ DD^i(u_{i-1}, v_{i-1}, u_i, v_i), (E_0(y_i, s_i, t_i, w_i, z_1, u_i, v_i) \cup E_1(y_i, s_i, t_i, w_i, z_2, u_i, v_i))).$$

The recursion is equivalent to the following set of rewritten rules, where the only recursive rule (T_1) is a normalized 3-chain rule.

- $(G_0) \quad R(x, y_0, w, t_0, z, u, v) :- E_0(x, y_0, w, t_0, z, u, v).$
 $(G_1) \quad R(x, y_0, w, t_0, z, u, v) :-$
 $\quad A(x, s_0, t_0), B(w, w_0), C(z), D'(u, v, u_0, v_0), T(y_0, w_0, s_0, t_0, u_0, v_0).$
 $(T_0) \quad T(y, w, s, t, u, v) :- E(y, w, s, t, u, v).$
 $(T_1) \quad T(y, w, s, t, u, v) :-$
 $\quad AB(y, w, y_1, w_1), BA(s, t, s_1, t_1), DD(u, v, u_1, v_1), T(y_1, w_1, s_1, t_1, u_1, v_1).$
 $(E) \quad E(y, w, s, t, u, v) :- E(y, s, t, w, z_1, u, v); E_1(y, s, t, w, z_2, u, v).$

The constants a and d in the query are pushed into t_0 in T of (G_1) , and the constant b is pushed into y_0 in T . Therefore, two magic sets on AB and BA can be derived based on the binding propagation, and the semi-naive evaluation can be performed on the derived magic sets together with DD and E relations to evaluate the recursion T . $\square$

Because of the scope of the paper is on matrix-based compilation, the concrete algorithm for normalization of linear recursions based on the compiled chain form is not presented here. It is left as an exercise to interested readers.

7. Strength and Limitations of the Matrix-Based Compilation Method

The matrix-based compilation method compiles complex linear recursions into highly regular compiled forms or, alternatively, normalized n -chain recursions, which provides precise chain connection information in relational expressions for detailed query analyses and optimization. Moreover, it provides extra strength on capturing some bindings which are difficult to be captured otherwise.

7.1. Capture of more bindings for efficient query evaluation

Regular binding propagation techniques, such as magic rule rewriting, can propagate sufficient binding information into recursive rules for many recursions. However, it may encounter difficulties for certain recursions.

Example 11. We examine the binding propagation of magic rule rewriting method [4, 31] for a query " $? - R(a, y, c)$." on the linear recursion defined by the rule set $\{(D_1), (E_0)\}$ where a and c are constants.

Following the binding propagation rules [31] and the Magic rule rewriting method [4, 6], the goal node is adorned as $R^{bf b}$. The bindings in the adorned goal node are propagated to the subgoal R in the body of the recursive rule, resulting in an adorned subgoal node $R^{f b f}$, which are in turn propagated to the subgoal R in the body of the recursive rule at the next expansion, resulting in $R^{f f f}$. Obviously, $R^{f f f}$ cannot propagate any binding information to the body of the rule. The adorned rules, (D_1) and (D_{12}) represent such binding propagation, where (D_1) is an adorned rule of (D_1) with respect to the adorned goal node $R^{bf b}$, (D_{12}) is an adorned rule *unifying directly* (without renaming variables or modifying adornments) with (D_1) at the second expansion.

$$(D_1') \quad R^{bf b}(x, y, z) :- A^{bf}(x, y), R^{fbf}(x_1, z, z_1), B^{bb}(x_1, z_1).$$

$$(D_{12}') \quad R^{fbf}(x_1, z, z_1) :- A^{fb}(x_1, z), R^{fff}(x_2, z_1, z_2), B^{bb}(x_2, z_2).$$

Since there is no binding information which can be passed further to the subgoal R in (D_{12}') , the Magic Set involves the entire relation A . Clearly, the binding propagation cannot reduce the set of data to be examined in the semi-naive evaluation.

We may reorder the subgoals in the body of the rule in search for better rewritten rules [31]. It is easy to find that although some reordering may avoid the generation of the uninstantiated subgoal R^{fff} , it will introduce uninstantiated subgoals, which requires iterative processing of some uninstantiated (entire) data relation.

However, based on the matrix expansion-based compilation, a highly-regular compiled form has been derived for this recursion as shown in (D_c) . Alternatively, a set of rewritten rules is derived which transforms the recursion into a normalized single-chain recursion. The binding " $z = c$ " can be propagated to every further expansion of this normalized single-chain recursion. Obviously, the query can be processed more efficiently by such binding propagation, which has been discussed in Example 9. $\square$

7.2. Limitations: some optimization problems

The matrix expansion-based compilation method provides a simple and efficient way to compile complex recursions into highly regular compiled forms. However, previous studies have shown that even tests for boundeness or one-sidedness for linear recursions are NP-complete or undecidable problems [24, 33]. Is there any conflict between our conclusion and the results in previous studies?

In fact, when a linear recursion is compiled into an n -chain recursion using the expansion-based compilation algorithm, it is possible that the compiled form can sometimes be further optimized and simplified to an m -recursion for some $m < n$ (it becomes a bounded recursion when $m = 0$). In this sense, the n -chain recursion obtained from the compilation process is an "at most" n -chain recursion. The conditions for bounded linear recursions presented in the expansion-based compilation algorithm are only sufficient conditions. In general, it is a difficult problem to further simplify the compiled recursion. We examine one such example.

Example 12. According to the expansion-based compilation method, the recursion $\{(A_1), (A_0)\}$ is compiled into (A_c') .

$$(A_1) \quad R(x) :- A(x, x_1), R(x_1).$$

$$(A_0) \quad R(x) :- A(x, x_1).$$

$$(A_c') \quad R(x_0) = \bigcup_{i=1}^{\infty} A^i(x_{i-1}, x_i).$$

However, a careful examination of the compiled form may discover that (A_c') is equivalent to a simpler formula (A_c'') , a formula derived from the exit rule only.

$$(A_c'') \quad R(x) = A(x, x_1).$$

This is because the formula generated by the second expansion, " $A(x, x_1), A(x_1, x_2)$ ", enforces a stronger restriction on the derivable values of x than the formula " $A(x, x_1)$ " and thus cannot produce any new answers. This is also true for further expansions. This phenomenon was first discovered by Naughton and Sagiv [24]. In fact, if the predicate in the exit rule were different from the nonrecursive predicate in the recursive rule, such as " $R(x) :- B(x, x_1)$ ", where A is not *contained* in B , the recursion should still be a typical single-chain recursion. However, A and B could consist of a sequence of conjunctive predicates. It is a well-known NP-complete problem to test the containment mappings for complex conjunctive queries [24, 31, 33]. $\square$

Clearly, the compilation technique developed in this study cannot detect the boundedness for the above cases. However, it is an NP-complete or undecidable problem to test for containment mapping and perform further optimization on the compiled forms [24, 33]. Fortunately, without such optimization, regular recursive query evaluation algorithms still terminate and derive all the answers for such recursions in deductive databases. The inability of performing such further optimization is a limitation of our technique. However, it does not forbid the successful application of our compilation and query evaluation techniques to most linear recursive query processing problems.

8. Conclusions

We have developed a matrix expansion-based compilation method which constructs a variable connection matrix, the V-matrix, for a linear recursion and simulates recursion expansions by matrix expansions. The regularity (stability and periodicity) of linear recursions can be discovered by V-matrix expansions. Based on such regularity, compiled forms in either bounded or highly regular chain forms can be automatically generated for complex linear recursions. Alternatively, a query-independent rule rewriting process can be applied to the compiled n -chain recursion to transform it into an equivalent normalized n -chain recursions.

The compilation of linear recursions into highly regular relational expressions facilitates the quantitative analysis of recursive queries and generation of efficient query evaluation plans. Moreover, some bindings which are difficult to be captured by other techniques can be captured naturally by the expansion-based compilation technique. Therefore, the automatic generation of compiled forms for linear recursions is a powerful tool for the analysis and evaluation of complex linear recursions in deductive databases.

The V-matrix expansion-based compilation method has been implemented on the UNIX system in Simon Fraser University. The software, the V-compiler, is written in C using LEX and YACC. It generates highly regular compiled forms for complex linear recursions as described in the paper.

Many database-oriented application examples belong to relatively simple recursions: recursions in chain-rules, such as ancestor, supervisor, same-generation, etc.

Nevertheless, there are many interesting application programs which involve complex recursions with arithmetic operations and function symbols, such as list processing, air-flight reservation, network traversing, inventory management, etc [30]. Interestingly, a recursion with function symbols can be transformed into its function-free counterpart by function-predicate transformation [13]. Thus, the compilation technique developed here can also be applied to the compilation and evaluation of complex linear recursions with function symbols in deductive databases.

Our study is confined to linear recursions. It is an interesting research issue to generalize the expansion-based compilation technique to other kinds of recursions, such as multiple linear recursions [12, 25], nonlinear recursions [18] and mutual recursions in deductive databases. Although some studies [9, 18, 25] have been performed on the compilation and efficient evaluation of such recursions, most of the studies are confined to relatively simple variable connections. More research should be performed on the compilation of such kind of recursions in complex variable connections.

Acknowledgements.

The work was supported in part by the Natural Sciences and Engineering Research Council of Canada under operating grant A-3723 and a research grant from Centre for Systems Science of Simon Fraser University. The authors would like to thank Ju Wu and Zhongmin Guo for their implementation of the matrix-based compilation techniques on the UNIX system. Also, the authors would like to express their thanks to Lawrence J. Henschen, Michael Kifer, Raghu Ramakrishnan, Kotagiri Ramamohanarao, Yehoshua Sagiv, Shalom Tsur and Carlo Zaniolo for their helpful discussions on the binding propagation problems.

References

1. R. Agrawal and P. Devanbu, Moving Selections into Linear Least Fixpoint Queries, *Proc. 4th Int. Conf. Data Engineering*, Los Angeles, CA, February 1988, 452-461.
2. A. Al-Sukairi and L. J. Henschen, Query Independent Compilation of Linear Recursions, *Proc. 1990 Int. Conf. Software Engineering and Knowledge Engineering*, Chicago, Illinois, June 1990, 177-182.
3. F. Bancilhon, D. Maier, Y. Sagiv and J. D. Ullman, Magic Sets and Other Strange Ways to Implement Logic Programs, *Proc. 5th ACM Symp. Principles of Database Systems*, Cambridge, MA, March 1986, 1-15.
4. F. Bancilhon and R. Ramakrishnan, An Amateur's Introduction to Recursive Query Processing Strategies, *Proc. 1986 ACM-SIGMOD Conf. Management of Data*, Washington, DC, May 1986, 16-52.
5. C. Beeri, P. Kanellakis, F. Bancilhon and R. Ramakrishnan, Bounds on the Propagation of Selection into Logic Programs, *Proc. 6th ACM Symp. Principles of Database Systems*, San Diego, CA, March 1987, 214-226.

6. C. Beeri and R. Ramakrishnan, On the Power of Magic, *Proc. 6th ACM Symp. Principles of Database Systems*, San Diego, CA, March 1987, 269–283.
7. F. Bry, Query Evaluation in Recursive Databases: Bottom-Up and Top-Down Reconciled, In W. Kim, J.-M. Nicolas and S. Nishio (eds.), *Deductive and Object-Oriented Databases*, North Holland, 1990, 25–44.
8. D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur and C. Zaniolo, The LDL System Prototype, *IEEE Trans. Knowledge and Data Engineering*, 2(1), 1990, 76–90.
9. J. Han and W. Lu, Asynchronous Chain Recursions, *IEEE Trans. Knowledge and Data Engineering*, 1(2), 1989, 185–195.
10. J. Han, Multi-Way Counting Method, *Information Systems*, 14(3), 1989, 219–229.
11. J. Han, Compiling General Linear Recursions by Variable Connection Graph Analysis, *Computational Intelligence*, 5(1), 1989, 12–31.
12. J. Han and L. Liu, Efficient Evaluation of Multiple Linear Recursions, *IEEE Trans. Software Engineering*, 17(12), 1991.
13. J. Han and Q. Wang, Evaluation of Functional Linear Recursions: A Compilation Approach, *Information Systems*, 16(4), 1991.
14. J. Han, Constraint-Based Reasoning in Deductive Databases, *Proc. 7th Int. Conf. Data Engineering*, Kobe, Japan, April 1991, 257–265.
15. L. J. Henschen and S. Naqvi, On Compiling Queries in Recursive First-Order Databases, *J. ACM*, 31(1), 1984, 47–85.
16. Y. E. Ioannidis, A Time Bound on the Materialization of Some Recursively Defined Views, *Proc. 11th Int. Conf. Very Large Data Bases*, Stockholm, Sweden, August 1985, 219–226.
17. Y. E. Ioannidis and R. Ramakrishnan, Efficient Transitive Closure Algorithms, *Proc. 14th Int. Conf. Very Large Data Bases*, Long Beach, CA, August 1988, 382–394.
18. Y. E. Ioannidis and E. Wong, Towards an Algebraic Theory of Recursion, *J. ACM*, 38(2), 1991, 329–381.
19. H. V. Jagadish, R. Agrawal and L. Ness, A Study of Transitive Closure as a Recursion Mechanism, *Proc. 1987 ACM-SIGMOD Conf. Management of Data*, San Francisco, CA, May 1987, 331–344.
20. B. Jiang, A Suitable Algorithm for Computing Partial Transitive Closures, *Proc. 6th Int. Conf. Data Engineering*, Los Angeles, CA, February 1990, 264–271.
21. M. Kifer and E. L. Lozinskii, On Compile-Time Query Optimization in Deductive Databases by Means of Static Filtering, *ACM Trans. Database Syst.*, 15(3), 1990, 385–426.
22. R. Krishnamurthy and C. Zaniolo, Optimization in a Logic Based Language for Knowledge and Data Intensive Applications, *Proc. Int. Conf. of Extending*

- Database Technology (EDBT'88)*, Venice, Italy, March 1988, 16–33.
23. J. F. Naughton, One-Sided Recursions, *Proc. 6th ACM Symp. Principles of Database Systems*, San Diego, CA, March 1987, 340–348.
 24. J. F. Naughton and Y. Sagiv, A Decidable Class of Bounded Recursions, *Proc. 6th ACM Symp. Principles of Database Systems*, San Diego, CA, March 1987, 214–226.
 25. J. F. Naughton, R. Ramakrishnan, Y. Sagiv and J. D. Ullman, Efficient Evaluation of Right-, Left-, and Multi-Linear Rules, *Proc. 1989 ACM-SIGMOD Conf. Management of Data*, Portland, Oregon, June 1989, 235–242.
 26. R. Ramakrishnan, Magic Templates: A Spellbinding Approach to Logic Programs, *Proc. Int. Conf. Logic Programming*, Seattle, WA, August 1988, 140–159.
 27. Y. Sagiv, Optimizing Datalog Programs, In J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, 1988, 659–698.
 28. H. Seki, On the Power of Alexander Templets, *Proc. 8th ACM Symp. Principles of Database Systems*, Philadelphia, PA, March 1989, 150–159.
 29. L. Sterling and E. Shapiro, *The Art of Prolog*, The MIT Press, 1986.
 30. S. Tsur, Applications of Deductive Database Systems, *Proc. IEEE COMCON Spring'90 Conf.*, San Francisco, CA, Feb. 1990.
 31. J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. 2*, Computer Science Press, Rockville, MD, 1989.
 32. J. D. Ullman, Bottom-up Beats Top-down for Datalog, *Proc. 8th ACM Symp. Principles of Database Systems*, Philadelphia, PA, March 1989, 140–149.
 33. M. Y. Vardi, Decidability and Undecidability Results for Boundedness of Linear Recursive Queries, *Proc. 7th ACM Symp. Principles of Database Systems*, Austin, TX, March 1988, 341–351.
 34. L. Vieille, Recursive Axioms in Deductive Databases: The Query/Subquery Approach, *Proc. 1st Int. Conf. Expert Database Systems*, Charleston, SC, April 1986, 179–193.
 35. C. Youn, L. J. Henschen and J. Han, Classification of Recursive Formulas in Deductive Databases, *Proc. 1988 ACM-SIGMOD Conf. Management of Data*, Chicago, IL, June 1988, 320–328.

Appendix: The Proof of Theorem 1

Theorem 1. *In a single-unit recursive rule of arity n , the expansion of its V-matrix terminates at or before the n -th iteration. That is, $S + T \leq n$, where S is the stable level and T is the period of the V-matrix.*

Proof.

Let x_i (where $1 \leq i \leq n$) be a distinguished variable at the i -th argument position in the head predicate of the recursive rule and $u_i^{(k)}$ (where $1 \leq k \leq n$) be a nondistinguished variable at the corresponding argument position of the recursive predicate in the body. The initial V-matrix can be represented as,

[0]	x_1	x_2	...	x_n
[1]	$\mu(x_1)$	$\mu(x_2)$	...	$\mu(x_n)$

where $\mu(x_i)$ is a mapping of x_i to the corresponding column of row [1] following the V-matrix initialization and expansion rules. The mapping results in either a set of distinguished variables derived from x_i or a nondistinguished variable $u_i^{(1)}$. Since the same matrix expansion rules apply to the expansions of other rows in the V-matrix, the element in the column i of row $[j]$ can be considered as obtained by applying the mapping rules to the same column of row $[j-1]$, i.e., $\mu(\mu^{j-1}(x_i)) = \mu^j(x_i)$.

The theorem can be proved by induction on n .

First, the theorem is valid when $n = 1$. This is because there are only two cases when $n = 1$: (1) $\mu(x_1) = x_1$, and (2) $\mu(x_1) = u_1^{(1)}$. Obviously, in case (1), $S = 0$ and $T = 1$; and in Case (2), $S = 1$ and $T = 0$. In both cases, $S + T = 1$. The theorem is valid.

Suppose the theorem is valid when $n = 1, 2, \dots, l$. We prove that the theorem is valid for $n = l + 1$. The proof is partitioned into two parts.

Part I: There are no U-connected distinguished variables in row [0]. It can be divided into two cases: *Case 1:* $\mu(x_i) \neq \mu(x_j)$, for any $i \neq j$ ($i, j = 1, 2, \dots, l+1$); and *Case 2:* $\mu(x_i) = \mu(x_j)$, for some $i \neq j$ ($i, j = 1, 2, \dots, l+1$). We prove each case as follows.

Case 1: Case 1 can be proved by considering the following three circumstances.

- (1) $\mu(x_i) = x_i$ for some i ($i = 1, 2, \dots, l+1$). It is easy to show by contradiction that the circumstance is impossible. Without loss of generality, suppose $i = 1$. That is, $\mu(x_1) = x_1$. We have,

[0]	x_1	x_2	...	x_{l+1}
[1]	x_1	$\mu(x_2)$	...	$\mu(x_{l+1})$

Since this is a single-unit recursive rule, there must exist some j ($j = 2, 3, \dots, l+1$) such that $\mu(x_j) = x_1$. This contradicts the condition $\mu(x_i) \neq \mu(x_j)$, for any $i \neq j$.

- (2) $\mu(x_i) = u_i^{(1)}$ for some i ($i = 1, 2, \dots, l+1$). Without loss of generality, suppose $i = 1$. That is, $\mu(x_1) = u_1^{(1)}$. Since this is a single-unit recursive rule, there must exist some j ($j = 2, 3, \dots, l+1$) such that $\mu(x_j) = x_1$. Suppose $j = 2$. For the same reason, there must exist some k ($k = 3, \dots, l+1$) such that $\mu(x_k) = x_2$. Suppose $k = 3$. Thus, we have rows [0] and [1] as follows.

[0]	x_1	x_2	x_3	$\dots$	x_{l+1}
[1]	$u_1^{(1)}$	x_1	x_2	$\dots$	x_l
[2]	$u_1^{(2)}$	$u_1^{(1)}$	x_1	$\dots$	x_{l-1}
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
[$l+1$]	$u_1^{(l+1)}$	$u_1^{(l)}$	$u_1^{(l-1)}$	$\dots$	$u_1^{(1)}$

Following the matrix expansion rules, we have row [$l+1$] as shown above where row [$l+1$] contains no distinguished variables. Thus, we have $S = l + 1$, $T = 0$, and $S + T = l + 1$.

- (3) $\mu(x_i) = x_j$ where $i \neq j$ for some i, j ($i, j = 1, 2, \dots, l+1$). Without loss of generality, suppose $i = 1$, and $j = 2$. Then what can $\mu(x_2)$ be? It cannot be x_2 nor $u_2^{(1)}$ as discussed before. Nor can it be x_1 unless $l = 1$ based on the single-unit matrix assumption and case 1 assumption $\mu(x_i) \neq \mu(x_j)$, for any $i \neq j$. So $\mu(x_2)$ must be x_h ($h = 3, \dots, l+1$). Without loss of generality, let $h = 3$. The initial matrix becomes

[0]	x_1	x_2	x_3	$\dots$	x_{l+1}
[1]	x_2	x_3	$\mu(x_3) = ?$	$\dots$	$\mu(x_{l+1})$

For the same reason, we have $\mu(x_3) = x_4$, $\mu(x_4) = x_5$, etc. Therefore, we have the following initial matrix.

[0]	x_1	x_2	x_3	$\dots$	x_l	x_{l+1}
[1]	x_2	x_3	x_4	$\dots$	x_{l+1}	x_1

By matrix expansions, row [$l+1$] $\equiv$ row [0]. Thus, $S = 0$, $T = l + 1$, and $S + T = l + 1$.

Case 2: Case 2 can be proved by induction on its sub-matrix. We show that the theorem is valid when $\mu(x_i) = \mu(x_j)$ for some $i \neq j$ ($i, j = 1, 2, \dots, l+1$). Without loss of generality, suppose $i = 1$, and $j = 2$. The matrix becomes,

[0]	x_1	x_2	x_3	$\dots$	x_{l+1}
[1]	$\mu(x_2)$	$\mu(x_2)$	$\mu(x_3)$	$\dots$	$\mu(x_{l+1})$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
[l + 1]	$\mu^{l+1}(x_2)$	$\mu^{l+1}(x_2)$	$\dots$	$\mu^{l+1}(x_i)$	$\dots$

Notice that at most l different distinguished variables are contained in row [1]. Assume that some distinguished variable x_j is absent. Then x_j will not appear in the V-sub-matrix headed by $[\mu(x_2), \mu(x_3), \dots, \mu(x_{l+1})]$. Ignore the column $\mu(x_i) = u_i^{(1)}$, if any. According to the assumption of induction for the V-submatrix, there exist two integers i and j ($1 \leq i < j \leq l + 1$) such that $\text{row}[i] = \text{row}[j]$, or there exists an integer k such that row $[k]$ contains no distinguished variables. Evidently, this is the same situation for original V-matrix. The case 2 is proved.

Part II: There are U-connected distinguished variables in row [0].

We show that the theorem holds for $n = l + 1$ when there are U-connected distinguished variables in row [0]. Without loss of generality, assume that x_1 is U-connected with x_2 .

[0]	x_1	x_2	x_3	$\dots$	x_{l+1}
	$\bullet$	$\bullet$			
[1]	$\mu(x_1)$	$\mu(x_2)$	$\mu(x_3)$	$\dots$	$\mu(x_{l+1})$

According to the matrix expansion rules, the distinguished variables x_1 and x_2 will always appear in the row $[i]$ ($i = 1, 2, \dots, l$) as in one set. The set variable $\{x_1, x_2\}$ can be served as a new distinguished variable.

Note that there are at most l different distinguished variables contained in row [1]. That is, either one of elements in row [1] is a nondistinguished variable (for example, $\mu(x_i) = u_i^{(1)}$) or there exist two elements such that $\mu(x_i) = \mu(x_j)$. When $\mu(x_i) = u_i^{(1)}$, the element can at most serve as a bridge (when it has U-connections with other nondistinguished variables as shown in D_1) to pass the distinguished variables to other elements. Once the pass is done, it will not play any role of binding passing in future expansions. When $\mu(x_i) = \mu(x_j)$, the two columns i and j can be treated as one in further expansions. Therefore, after the second expansion, the first row (row[0]) and the i th column can be deleted from the V-matrix, which results in a V-submatrix. That is, the row [1] in the original V-matrix can serve as the 0-th row of the submatrix, and $\{x_1, x_2\}, x_3, \dots, x_{l+1}$ can be treated as l new distinguished variables. According to the induction assumption, the new V-submatrix will be in the situation that either a row repeats some previous rows or the whole row contains no distinguished variables at no more than l expansions. Thus, the original matrix will have the same situation at no more than $l + 1$ expansions. Therefore, the theorem is proved. $\square$

Centre for Systems Science

• Simon Fraser University established the **Centre for Systems Science** as a provincial centre for advanced study. The objective of CSS is twofold: coordinate and combine advanced systems research at SFU, and encourage application of research results within industry in order to further the technological development of British Columbia.

• CSS has assembled three research groups to streamline its efforts. The orientation of these groups is as follows: intelligent systems, microelectronics, and computer and communications systems. Because of shared research interests, this latter group merges with LCCR (**Laboratory for Computer and Communications Research**) for the publication of technical reports.

• CSS's three groups involve over 80 skilled faculty researchers principally from SFU's Schools of Engineering Science and Computing Science, but also draw substantially on Mathematics, Physics, Geography, Philosophy, Linguistics, Psychology, Communication, and Kinesiology.